Bölüm 02

Visual Studio Ortamı

Visual Studio

.NET Framework

Visual C# Arayüzü ile Program Yazma, Derleme ve Koşturma

Sıkça Sorulan Sorular

Visual Studio

Birinci Bölümde, kaynak programı, derlemeyi, gerektiğinde programı düzeltmeyi (debug) ve derlenen programı koşturmayı (run) öğrendik. Bu işleri yaparken basit bir editör ile C# derleyicisine (cs.exe) gereksememiz oldu. Programı derleme ve koşturma eylemlerini konsoldan yazdığımız komutlarla gerçekleştirdik. Hepsi çok yalın ve kolay işlerdi. O yaptıklarımız hemen her dildeki program için yapılanlardır ve işin özüdür. Programcılığa başlayan herkes o süreci bilmek zorundadır.

Öte yandan, o yalın eylemler, gerekseme duyduğumuz tekerleği kendi ellerimizle yapmak gibidir. Her zaman mümkündür; ama en iyi yol değildir. Birileri bizim için tekerleği yapmışlarsa, onu hazır alıp arabamızı yürütmek daha akıllıca olabilir.

Günümüzde programlama işi biraz tekerlek yapmaya benzedi. Mükemmel hazır tekerleri alıp kullanıyormuşçasına kaynak programı yazmayı, derlemeyi ve koşturmayı çok çok kolaylaştıran araçları alıp kullanabiliriz. Bu tür araçlara bütünleşik (integrated) program, kullanıcı dostu program, arayüz gibi adlar verilir. Hemen her dilde bu tür araçlar bolca üretilmiştir ve programcının emrindedir.

C# için de programlamanın evrelerini kolaylaştıran araçlar vardır. Biz, bu derste Microsoft'un geliştirdiği *Visual Studio 2008* bütünleşik programını (arayüz) kullanacağız. Bunun iki sürümü vardır. Birisi profesyonel programcılar için lisanslı olan sürümüdür. Ötekisi, programlamaya yeni başlayanların bütün gereksemelerini karşılayacak nitelikteki serbest sürümüdür. Ücretsizdir, lisans almaya gerek yoktur ve internetten indirilebilmektedir.

Visual Studio 2008, Microsoft'un C++, C#, VB, J# , Jscript dillerinde program yazmayı ve derlemeyi kolaylaştırmak için hazırladığı çok amaçlı bir arayüzdür.

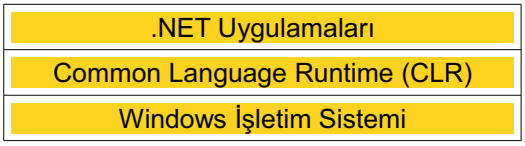
.NET Framework

Programlamaya yeni başlayanları gereksiz ayrıntıyla boğmamak için, *Visual Studio*'nun iç yapısını açıklamaya girmeyeceğiz. Onu bir otomobil gibi görelim, motorunun kaputunu hiç açmadan direksiyonuna oturalım ve kumanda tablosundaki düğmelerin işlevlerini öğrenelim. Çünkü biz başlangıçta motor yapımcısı değil, araba sürücüsü olmak istiyoruz. Sürücü belgesi bize uzun zaman yetecektir. Epeyce zaman sonra zaten merak ederek motorun kaputunu açıp, onun içindekileri görmek isteyeceğiz. Ancak, şu kadarını söylemek hem yararlı hem gereklidir. Microsoft *C++, C#, VB, J#* gibi dilleri derlemek ve çalıştırmak için *.NET Framework* denilen ortak bir platform yarattı. C# kaynak programları kendiliğinden o ortama taşınır, orada derlenirler, orada çalışırlar. O nedenle, C# dilini öğrenirken, ayrı bir çaba harcamadan ve çoğunlukla farkında bile olmadan *.NET Framework* yapısını da epeyce öğrenmiş olacağız.

.Net Framework genel amaçlı bir program geliştirme platformudur. 2000 yılında Microsoft firması tarafından ilk sürümü ortaya konmuştur. Bu kitap yazılırken, 3.5 sürümü piyasaya çıkmış durumdadır. Zamanla yeni sürümlerinin çıkması doğaldır. İşlevleri bakımından *java platformu* ile aynıdır. Her ikisiyle istemcide, sunucuda, mobil ya da gömülü cihazlarda uygulama programları geliştirilebilir, web uygulamaları, veritabanı uygulamaları gibi bir çok iş yapılabilir. Her ikisi de işletim sistemine ve donanıma bağlılığı ortadan kaldırmayı hedeflemektedir.

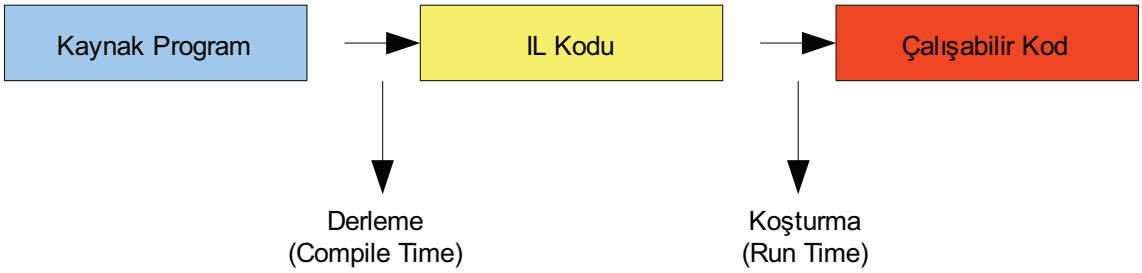
Bunu yapabilmek için, java, bellekte *virtual machine (VM)* denilen sanal bir makina yaratır. Java derleyicisi kaynak programı *bytecode* denilen ve *VM* üzerinde çalışabilen bir tür sanal makina diline dönüştürür. Böylece platformdan bağımsızlık sağlanır.

Microsoft'un uygulama geliştirme dilleri olan *C++, C#, VB, J#* ile yazılan kaynak kodlar derlenince *IL (Intermediate Language)* koduna dönüşürler. Bu dönüşüm, java'nın *bytecode*'una benzetilebilir. Sonra *JIT (Just-in-Time)* derleyicisi koşturulacak *IL* kodunu makine diline dönüştürür. *IL* bazı kaynaklarda *MSIL (Microsoft Intermediate Language) veya CIL (Common Intermediate Language) diye adlandırılır. CLR (Common Language Runtime)* ise *IL* koduna dönüşen programların koştuğu yerdir.



Başka bir deyişle, *C++, C#, VB, J#* dillerinden biriyle yazılan bir program derlenince doğrudan *Windows İşletim Sistemi* üzerinde çalışmaz; onun üzerinde kurulan sanal *CLR* üzerinde çalışır. Çünkü *C++, C#, VB, J#* dillerinde yazılan programlar derlenince, Windows İşletim sisteminde çalışabilen (executable) kodlara değil, *IL* diline dönüşür. Bunun ne anlama geldiğini, basite indirerek, şöyle açıklayabiliriz. C# dilinde `int` tipi bir değişken tanımladığımızda, derleyici onu *.NET Frameworkta'ki* karşılığı olan `Int32` tipine dönüştürür. Bunun gibi, kaynak programı hangi dilde yazarsanız yazınız, oradaki her şey *.NET Framework'taki* karşılığına dönüşür. Genelleştirirsek şöyle diyebiliriz. *C++, C#, VB, J#* dillerinin veri tipleri .NET'in *CTS* (Common Type System) denilen veri tiplerine dönüşür. Böylece bu diller için ortak bir platform yaratılmış olmaktadır. C# dilinin veri tiplerinin .NET 'deki karşılıklarını *Veri Tipleri ve Değişkenler* bölümünde ayrıntılı olarak ele alacağız.

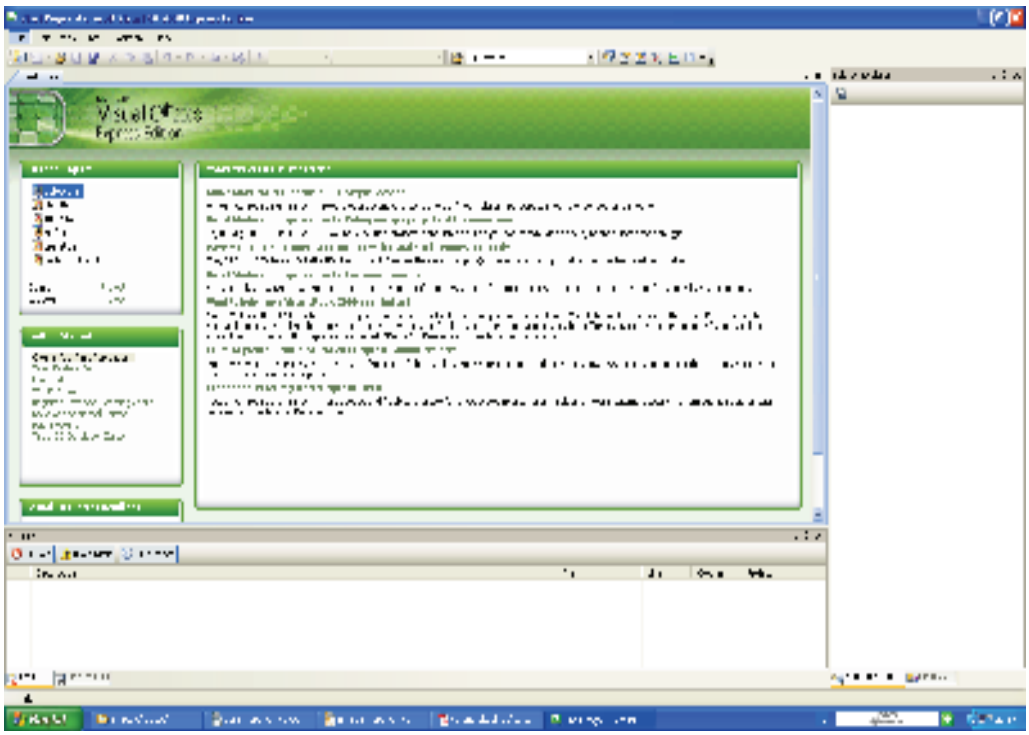
.NET Framework çok geniş bir kütüphaneye sahiptir. Onunla ilişkili olarak *CLI* (Common Language Infrastructure), *CTS* (Common Type System), metadata, *VES* (Virtual Execution Environment), *CLS* (Common Language Specification) gibi kavramları duyabilirsiniz. Ama onları şimdilik gözardı edebilirsiniz. Çünkü biz, Visual Studio ve .NET 'in iç yapısıyla değil, kumanda odasıyla ilgili olacağız.



Bu kısa açıklamadan sonra, Visual Studio ile çalışmaya başlayabiliriz. Bilgisayarınızda *Visual Studio 2008* 'in içinde ayrı bir modül olan *Microsoft Visual C# 2008 Express Edition* 'in yüklü ve çalışır durumda olduğunu varsayıyoruz. Nasıl yükleneceğini internetten indirdiğiniz yerden okuyabilirsiniz.

Visual C# Arayüzünü Başlatmak

Başlat **->** *Programlar -> Microsoft Visual C# 2008 Express Edition* sekmesini tıklayın. Karşınıza şuna benzer bir pencere gelecektir.

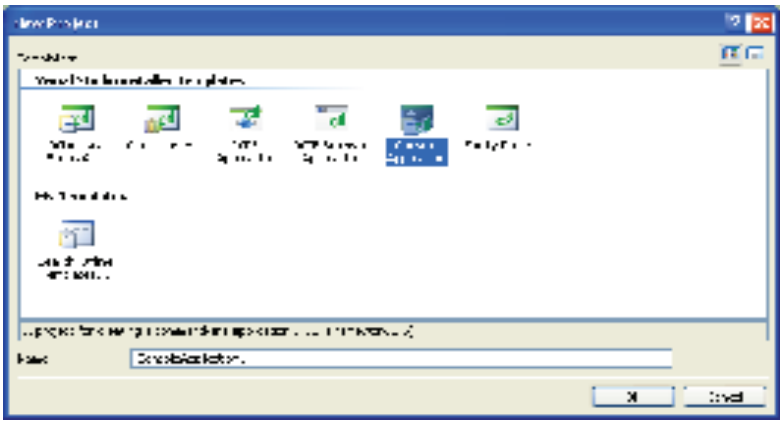


Sol üstteki `Recent Projects` bölümü sizde farklı olacaktır.

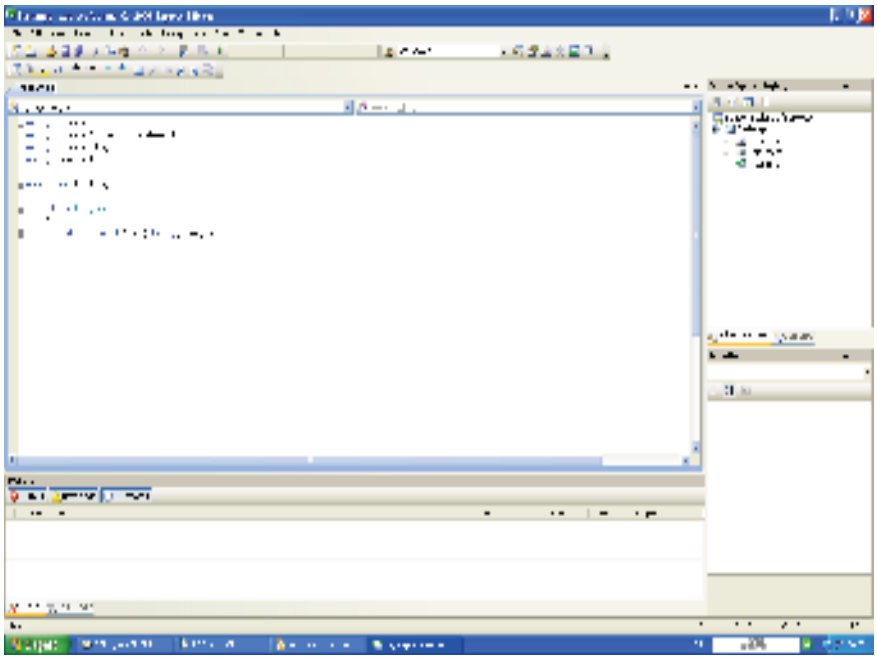
Visual Studio ilk açılışında projeleri kaydedeceği dizini sorar. O anda istediğiniz sürücüde istediğiniz bir dizin yaratabilir veya Visual Studio'nun önerdiği öndeğer (default) proje dizinini kabul edebilirsiniz. Bu

kitabın yazıldığı makinada c:\vsProject adlı bir dizin yaratılmıştır. Bütün projeler oraya kaydedilmektedir.

Üst satırda menü, onun altında araçların yer aldığı satır yer alır. Onun altındaki satırın solunda *Start Page*, sağında *Solution Explorer* sekmelerini göreceksiniz. *Start Page* sekmesine sağ tıklayınız ve açılan pencerede *Close* 'u tıklayınız. Görüntüdeki pencere kapanacaktır. Sonra File -> New Project 'i tıklayınız. Görüntüye *New Project* penceresi gelir:



Bu pencerede bize altı seçenek sunuluyor. İlk işlerimiz konsol uygulamaları olacaktır. C# ile nesne programlamayı onunla öğreneceğiz. Ondan sonra öteki uygulamaları kolayca yapabilir duruma geleceğiz. Pencereden *Console Application* 'i tıklıyoruz. Onun ikonu seçili duruma gelir. Sonra, pencerenin altındaki *Name* alanına gidiyoruz. Bu alan, yapacağımız konsol uygulamasının hangi aduzayına (*namespace)* ait olduğunu belirliyor. *Name* alanında *ConsoleApplication1* adı yazılıdır. Olduğu gibi bırakabiliriz ya da istediğimiz başka bir ad verebiliriz. *Başlangıç* diyelim ve OK düğmesine basalım. Karşımıza şu pencere çıkar:



Sol üstte *Başlangıç.Program* sekmesi altında Visual Studio 'nun editör pencesi görünüyor. Bu pencerede,

kaynak program için Visual Studio bazı temel kodları hazır yazılmış olarak önümüze koyuyor.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Başlangıç
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Bu programın ilk dört satırının her birisindeki *using* sözcüğü bir *namespace (aduzayı)* çağırıyor. İlke olarak, çağrılan bir ad uzayındaki sınıflar, o sınıfların içindeki değişkenler ve metotlar sanki bu programın içinde tanımlanmış gibi kullanılabilir (kısıtları ayrıca göreceğiz). İstediğimiz zaman, `using` anahtar sözcüğü ile başka aduzayları da çağırabiliriz. İlk derslerde, yalnızca *System* aduzayına gereksememiz olacak. O nedenle, sonraki üç satırı silebiliriz. Silmesek de bize bir zarar vermezler.

Visual Studio, bize yaptığı program önerisinde aduzayının (`namespace`) adının '*Başlangıç'* olduğunu belirtiyor. Zaten bu adı biz vermiştik. Şimdi yazacağımız programdaki sınıf veya sınıflar, *Başlangıç* aduzayına ait olacaklardır. Bu aidiyet, fiziksel kayıt ortamında değil, mantıksal (logic) olarak yapılır. Dolayısıyla, biz programlarımızı istediğimiz dizin veya alt dizine kaydedebiliriz.

Hemen belirtelim ki, yazdığımız her program için bir namespace (aduzayı) belirlemek zorunda değiliz. Programa namespace blokunu koymadığımız zaman, yarattığımız sınıflar hiç bir aduzayına ait olmazlar; ama kendi başlarına çalışabilirler; başka aduzaylarını çağırabilirler.

Programın sonraki satırlarına bakarsak şunu görüyoruz. *Visual Studio*, bize sınıf (class) adı olarak *Program* öneriyor ve hemen o sınıfın içinde `Main()` metodunun imzasını atıyor. Sınıf adını aynen bırakabilir veya istediğimiz bir ad ile değiştirebiliriz. Örneğin, *Giriş* diyelim. İlk programımızda *Visual Studio* ile kaynak programımızı yazmayı ve derleyip koşturmayı öğrenmek istiyoruz. Onun için, basit bir deyim yazmakla yetineceğiz. Gelenek olduğu gibi, konsola `"Merhaba C#"` yazdıralım. Bunu yazdırabilirsek, aynı yöntemle bir roman bile yazdırabileceğimizi biliyoruz. Bunu yapmak için, önceki bölümde gördüğümüz gibi, `Main()` metodunun gövdesine `Console.Writeline( "Merhaba C# " )` deyimini eklememiz yetecektir. Böylece, *Visual Studio* 'nun editör penceresinde programımız şu biçimi alır.

```
using System;

namespace Başlangıç
{
    class Giriş
    {
        static void Main(string[] args)
        {
            Console.WriteLine( "Merhaba C# " ) ;
```

```
            }
        }
}
```

Şimdi programımızı derleyebiliriz. Ama ondan önce yazdıklarımızı kaydetmekte yarar var. Ekranın sağında yer alan *Solution Explorer – Başlangıç* penceresindeki *Program.cs* Visual Studio 'nun önerdiği addır. Bu adı koruyabiliriz veya istiyorsak değiştirebiliriz.

C# programları `.cs` uzantısı alır ve sınıfın adını almak zorunda değildir. C# buna aldırmaz. Ama içinde bir tek sınıf olan programlarda sınıf adı ile program adını aynı yaparsak, çağrışımda bize kolaylık sağlar. Nasıl yapıldığını görmek için, *Solution Explorer – Başlangıç* penceresindeki *Program.cs* adına sağ tıklayalım, açılan penceredeki *Rename* sekmesini seçerek, yeni adı girelim. Sınıfın adını taşısın diye, programın adını *Giriş.cs* olarak değiştirelim. İsterseniz başka ad da verebilirsiniz. Sonra *File -> Save Giriş.cs* seçeneğini tıklayarak, programı kaydedelim. Aynı anda birden çok programla çalışabiliriz. Hepsini kaydetmek için *File -> Save All* seçeneğini tıklarız. Aynı işi, araç satırındaki

düğmeleriyle de yapabiliriz. Soldaki düğme, etkin olan penceredekini kaydeder. Sağdaki düğme, bütün programları kaydeder. Artık programımızı derleyelim.

Menü satırında *Debug _> Start Without Debugging* sekmesine tıklayınız. İsterseniz onun yerine `Ctrl+F5` tuşlarına da basabilirsiniz. Konsol açılır ve üzerinde *Merhaba C#* görünür:

Böylece, *Visual Studio* ile ilk programımızı yazdık, derledik ve koşturduk. Bundan sonrakiler de bu kadar basit olacaktır. Şimdi *Visual Studio* 'nun bazı hünerlerini görelim.

En sondaki '}' parantez simgesini kaldırıp programı derlemeyi deneyelim. Derleyiciden şu uyarı mesajını alırız.

Error	Description	File	Line	Column	Project
1	} expected	C:\vsProjects\Başlangıç\Başlangıç\Giriş.cs	11	6	Başlangıç

Program yazdıkça, benzer mesajları sürekli alıyor olacağımız için, bu mesajın nasıl okunacağını bir kez açıklamakta yarar olacaktır:

`Error` sütunu programda derleyicinin bulduğu hataların sayısı verilir. Bizim programda 1 hata olduğu işaret ediliyor.

`Description` sütununda derleyicinin gördüğü hatanın açıklaması verilir. Örneğimizde '`} expected`' dendiğine göre, programımızda ' } ' simgesinin eksik olduğu belirtiliyor.

`File` sütününda hatanın hangi dosyaya ait olduğu belirtilir.

`Line` sütununda, hatanın kaçıncı satırda olduğu belirtilir. Örneğimizde, hatanın 11-inci satırda görüldüğü belirtiliyor.

`Column` sütununda hatanın hangi kolonda oluştuğu belirtililir. Örneğimizde, hatanın 6-ıncı kolonda oluştuğu işaret ediliyor. [Kullandığımız ekranın çözünürlüğüne bağlı olarak, bizim saydığımız kolon numarası, derleyicinin saydığı kolon numarasından farklı olabilir. Ama derleyici, hatayı gördüğü yere küçük kırmızı dalga simgesi koyar. Hatayı orada aramalıyız.]

`Project` sütününda programın hangi aduzayına (namespace) ait olduğu belirtilir. Örneğimizde `C:\vsProjects\Başlangıç\Başlangıç\Giriş.cs` dosyası yazılıdır. Burada yazılanlar soldan sağa doğru şunlardır: dosyanın kaydedildiği sürücü adı (`C:`), dizin adı (`vsProject`), aduzayı (`Başlangıç`), sınıf adı (`Başlangıç`) ve dosya adı (`Giriş.cs`).

Hatanın işaret edildiği yere imleci götürüp '}' simgesini yazalım. Sonra, başka bir hata yapalım. Örneğin, gövde içindeki deyimin sonundaki (;) simgesini kaldıralım. Bu kez derleyicimiz hatanın 10-uncu satır, 45-inci sütunda (;) eksikliğinden kaynaklandığını belirten hata mesajını iletecektir. Ayrıca hatanın oluştuğu yere kırmızı renkli küçük bir dalda simgesi koyacaktır.

Error	Description	File	Line	Column	Project
1	; expected	C:\vsProjects\Başlangıç\Başlangıç\Giriş.cs	10	45	Başlangıç

Blok Girintileri

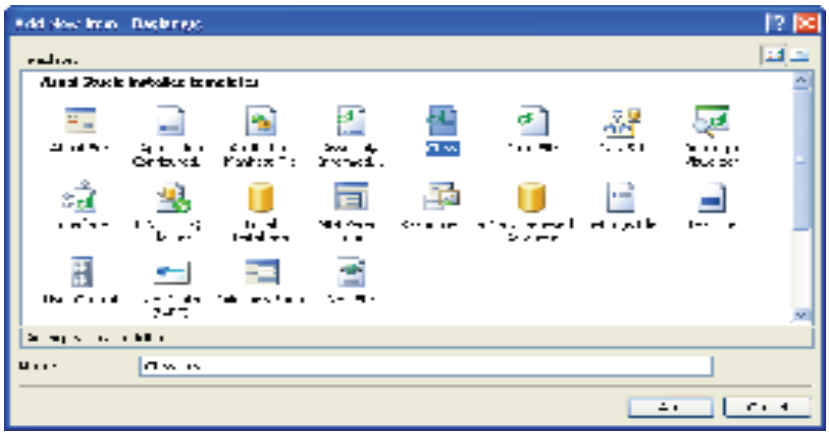
Yeni bir deneme daha yapalım. Programın yazılış biçemindeki blok girintilerini yok edelim:

```
using System;

namespace Başlangıç{
class Program    {
static void Main(string[] args)
        {
            Console.WriteLine("Merhaba C# ");
}
}
}
```

Bu aşamada `Ctrl+E+D` tuşlarına basınız. Programda blok girintilerinin yeniden oluştuğunu göreceksiniz. `Visual Studio` ile çalışırken, programınızda blok girintileri kendiliğinden oluşur; onları yapmak için zaman ve emek harcamanıza gerek yoktur. Ancak, programda söz dizimi hatası varsa, hata giderilene kadar otomatik biçemleme yapılmaz.

Şimdi, Visual Studio ortamını biraz daha yakından tanıyalım. Ekranın sağındaki Solution Explorer penceresinde Başlangıç aduzayını sağ tıklayalım. Açılır pencerelerden Add -> Class sekmesine tıklayalım.



Önümüze Add New Item adlı şablonlar penceresi gelir. Bu pencerede C# ile yapılabilecek işlerle ilgili şablonlar yer alır. Bu şablonlar içinden Class ikonunu seçelim. Pencerenin altında Name alanında Class1.cs öndeğeri (default vakue) yer alır. Bu öneriyi kabul edebiliriz. Ama yapacağımız işi çağrıştıran bir ad vermek daha uygun olur. Class1.cs yerine Toplama.cs yazalım. Önümüze editör penceresi gelecektir. Editörle aşağıdaki programı yazalım.

Toplama.cs

```
using System;
namespace Başlangıç
{
    class Toplama
    {
        static int x = 3;
        static double y = 14.5;

        static double Topla()
        {
            short z = 5;
            return x + y + z;
        }
    }
}
```

Blok girintileri düzgün değilse Ctrl+E+D tuşlarına basarak, blokları düzgünleştirebiliriz. Yazdığımız her programı satır satır çözümlemeyi; yani her deyimin ne iş yaptığını anlamayı alışkanlık edinmeliyiz. Öğrenme sürecinde bu iyi bir alışkanlık olacaktır. Bu programı çözümlersek şunları göreceğiz. Başlangıç aduzayına ait Toplama adlı bir sınıf ile bu sınıfın içinde Topla() adlı bir metot tanımlanıyor. Sınıfın `x` ve `y` adlı iki değişkeni var. Sınıf içinde tanımlanan değişkenler ile metotlara `sınıfın öğeleri (class members)` denilir. Topla sınıfının üç öğesi var. Değişkenlere, çoğunlukla, veri alanı ya da kısaca alan (field) denilir. Bu kitapta değişken, veri alanı ve alan sözcüklerini eş anlamlı kullanacağız.

```
static int x = 3;
```

bildirimi `int` veri tipinden `x` adlı bir değişken tanımlandığını, bu değişkene ilk değer olarak `3` atandığını, değişkenin `static` olduğunu bildirir. `static` nitelemesi, daha önce de söylediğimiz gibi, Toplama

sınıfının bir nesnesini yaratmadan `x` değişkenine erişilebileceğini belirtir. `static` nitelemesini ileride daha ayrıntılı inceleyeceğiz. *Değişkene erişmek* demek, istendiğinde ona değer atamak, istendiğinde atanmış değeri okuyup işlemlerde kullanabilme yeteneğine sahip olmak demektir.

```
static double y = 14.5;
```

deyimini yukarıdaki gibi çözümlenebilir.

```
public static double Topla()
```

deyimi değer kümesi `double` olan `Topla()` metodunun bildirimidir. Bu bildirimde metodun adı, değer kümesi ve varsa parametreleri yazılır. Bunlar bir metodu kesinkes belirtirler. O nedenle

```
double Topla()
```

ifadesine metodun imzası denilir. Metotlar imzalarıyla birbirlerinden ayırt edilirler. Adları, değer kümeleri veya parametrelerinden enaz birisi farklı ise iki metot farklı olur. Örneğin, adları ve değer kümeleri ve hatta işlevleri aynı olan iki metot parametreleri farklı olduğu için birbirlerinden ayırt edilebilirler.

Metot imzasının önünde

```
public static
```

nitelemelerini görüyoruz. Bunlardan `static` nitelemesi önce söylediğimiz işleve sahiptir. `public` nitelemesi ise, metoda dışarıdan erişimi; yani onun başka sınıflardan çağrılmasına izin veren bir erişim belirtecidir. *Erişim Belirteçleri* 'ni ileride ayrıntılı göreceğiz.

Son olarak `Topla()` metodunun gövdesine bakalım. `short` tipinden `z` adlı bir değişken bildirilmiş ve ona `5` değeri atanmıştır. Bu şekilde metodun gövdesinde tanımlanan değişkenlere metodun `yerel (local)` değişkeni ya da iç değişkeni denir. Metodun alacağı değer

```
return x + y + z;
```

deyimi ile belirtilmiştir. Bunun anlamı apaçıktır, fonksiyonun değeri `x, y ,z` değişkenlerinin değerlerinin toplamıdır. Metodun değeri daima `return` anahtar sözcüğü ile belirtilir. `return` ifadesinden çıkan değer metodun *değer kümesine* ait olmalıdır. Bu örnek için söylersek, `x+y+z` toplamı, metodun değer kümesi olan `double` veri tipinden olmalıdır. Aksi halde derleme hatası doğar.

Artık `Toplama.cs` programımız tamamdır. `File -> Save Toplama.cs` sekmesini tıklayarak ya da `CTRL+S` tuşlarına basarak dosyamızı kaydedelim. Ekranın sağ yanındaki `Solution Explorer` penceresindeki `Başlangıç` aduzayı altında `Toplama.cs` programını görüyor olmalıyız.

Şimdi aynı yolları izleyerek aşağıdaki iki programı yazalım ve kaydedelim.

Çıkarma.cs

```
using System;

namespace Başlangıç
{
    class Çıkarma
    {
        static double x = 17;
        static double y = 8.5;

        public static double Çıkar()
        {
            return x - y;
```

```
        }

    }
}
```

Ana.cs

```
using System;

namespace Başlangıç
{
    class ana
    {
        static void Main(string[] args)
        {
            double y = Toplama.Topla();
            double x = Çıkarma.Çıkar();
        }
    }
}
```

`Ana.cs` programını çözümleyelim. Ana sınıfında

```
static void Main(string[] args)
```

bildirimi ile `Main()` metodu tanımlanmıştır. `Main()` metodu genellikle bu biçemde bildirilir. `Main()` metodunun `String[]` tipinden `args` adlı bir array parametresi var. Bu parametreyi çoğunlukla kullanmayacağız. Kullanmayacağımız zaman parametreyi yazmasak da olur. Ama yazmanın sakıncası yoktur. `Main()` metodu bir programı çalıştıran asıl metot olduğu için, ona ayrı bir özen gösterelim ve yukarıda gösterildiği gibi parametresini yazmayı alışkanlık edinmeye çalışalım. Main() 'in gövdesindeki deyimleri, sırasıyla, nceleyelim.

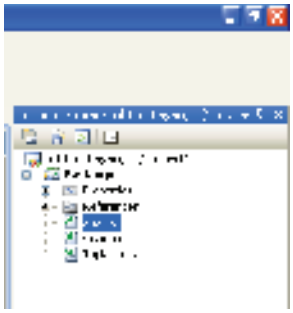
```
Toplama.Topla();
```

deyimi `Toplama` sınıfı içindeki `Topla()` metodunu çağırıyor. Bu eyleme *metot çağırma* veya *mesaj* iletme diyoruz. Bu mesajla, bir sınıftan başka bir sınıfa ait metodu çağırıp işlevini yapmasını istiyoruz. Bu mesajlaşma eylemi, nesne programlamanın (object oriented programming) üstün bir niteliğidir.

İkinci deyim olan

```
Çıkarma.Çıkar();
```

deyiminin açıklaması da benzerdir. Şimdi sağ taraftaki Solution Explorer penceresinde Başlangıç aduzayına ait olarak yazdığımız `Ana.cs, Çıkarma.cs` ve `Toplama.cs` programlarını görüyoruz.



Bunlardan herhangi birisine çift tıklarsak, editör penceresine onun kaynak programı gelir. Önümüze gelen programda, istediğimiz değişikliği yapabiliriz.

Program seçme işini editör penceresinden daha kolay yapabiliriz. Bencerenin üstündeki programlar satırında `Ana.cs, Çıkarma.cs` ve `Toplama.cs` programlarını görebiliriz. Onlardan birisine tıklarsak, o program editörde açılır. Her üç programı tam yazdığımıza emin olunca, `File -> Save All` sekmesine tıklayıp her üçünü birden kaydedelim. Hepsini birden kaydetmenin ikinci bir yolu CTRL+Shift+S tuşlarına basmaktır. Ama en kolayı, ekranın üstten üçüncü satırında yer alan sıralanmış disketler simgesine tıklamaktır. Tek disket simgesine tıklanınca, o anda editörde etkin olan dosya kaydedilir. Disketler grubuna tıklayınca, bütün dosyalar birden kaydedilir.

Programlarımızı koşturmadan önce, Visual Studio'nun birkaç aracını daha görelim. Örneğin, Ana.cs yi tıklayıp açalım.

Imleci herhangi bir stırın herhangi bir yerine koyalım. Sonra üstten üçüncü satır içindeki 5-inci ve 6-ıncı girintileme simgelerine [girinti artırıp azaltan (increase/decrease indent)] tıklayarak, girintilerin nasıl azalıp çoğaltıldığına bakalım. Eğer, Visual Studio'nun otomatik yaptığı blok girintilerinden memnun değilseniz, bu düğmeleri kullanabilirsiniz.

Tekrar imleci herhangi bir stırın herhangi bir yerine koyalım. Sonra üstten üçüncü satır içindeki 7-inci ve 8-inci simgelere [açıklama koy/kaldır (comment/uncomment)] tıklayarak, satırlara nasıl açıklama simgesi (//) konulduğuna bakalım.

Visual Studio'nun penceresine istediğiniz alt pencereleri getirebilirsiniz. Bunun için `View` açılır penceresinden istediğinizi seçiniz. Özellikle, programın koşması sırasında oluşabilecek hataları görmek için `Error List` penceresini açmalısınız.

Visual Studio'nun penceresindeki öteki simgelerin işlevlerini zamanla öğreneceksiniz. Ama çok merak edenler, `Help` sekmesinden hemen bilgi alabilirler.

Bunca sabırdan sonra, artık programlarımızı koşturmayı deneyebiliriz. Ne yapacağımızı zaten biliyoruz. `Ana.cs` programına tıklıyoruz. `Main()` metodu burada olduğu için, program girişi bu sınıftan olacaktır. Her programda bir tek giriş yeri olabilir. Birden çok `Main()` programı olsa bile, onlardan yalnızca bir tanesi giriş yeri olmalıdır.

`Ana.cs` programını çalıştırınca, onun içindeki `Main()` metodu, sırasıyla, `Toplama` ve `Çıkarma` sınıflarındaki `Topla()` ve `Çıkar()` metotlarını çağıracaktır. Dolayısıyla, yapacağımız iş, Ana.cs programını derleyip koşturmaktan ibarettir. Bunun için

```
Debug -> Start Without Debugging
```

sekmesine tıklamak ya da CTRL+F5 tuşlarına basmak yetecektir. Bunu yapınca ekrana siyah konsolun geldiğini ve içinde Hiçbir hata iletisi olmadığını göreceğiz. Öte yandan, Visual Studio ekranının altındaki Error List penceresinde de bir uyarı görülmüyor. Bu demektir ki, programlarımız derleme hatası (compile time error) vermemiştir; söz dizimleri doğrudur. Ayrıca koşma hatası (runtime error) da yoktur.

Ama konsolda programın yaptığı toplamayı ve çıkarmayı göremedik. Bunun nedeni apaçıktır. İstediklerimiz yapıldı, ama onları konsolda göremedik, çünkü konsola bir şey yazılmasını istemedik. Şimdi bu eksikliği kolayca giderebiliriz. Ana.cs programını açıp, Main() 'in gövdesindeki iki deyimi şöyle değiştirelim:

```
            Console.WriteLine(Toplama.Topla()) ;
            Console.WriteLine(Çıkarma.Çıkar()) ;
```

Bu değişikliği yapınca Ana.cs programı şu şekli alır.

Ana.cs

```
using System;

namespace Başlangıç
{
    class ana
    {
        static void Main(string[] args)
        {
            Console.WriteLine(Toplama.Topla());
            Console.WriteLine(Çıkarma.Çıkar());
        }
    }
}
```

Şimdi bunu koşturursak, konsola

```
22,5
8,5
```

yazıldığını göreceğiz.

Visual Studio Express Editions

Visual Studio Express Editions Microsoft'un Visual Studio ve SQL Server adlı ürünlerinin bir uzantısı sayılır. Öğrenciler, yeni öğrenmeye başlayanlar ve hatta programcılıkla bir hobby olarak uğraşanların dinamik Windows Uygulamaları yapmalarını, Web siteleri kurmalarını ve veritabanı uygulamaları yapmalarını kolaylaştıran bir araç, görsel bir arayüzdür. Ücretsizdir. İnternetten indirilebilir.

Visual Studio Express Editions şu bileşenlerden oluşur.

- Visual Basic 2008 Express Edition – İlk başlayanların tercih ettiği Visual Basic dili ile program geliştirme aracı
- Visual C# 2008 Express Edition – Windows uygulamaları için mükemmel bir araç. Tamamen Nesne Yönelimli bir dildir.
- Visual C++ 2008 Express Edition – C++ dilini bilenlerin tercih ettiği bir yarış atıdır.
- Visual Web Developer 2008 Express Edition – Dinamik web uygulamaları geliştirmeye yarayan kolay kullanılır bir araç.
- SQL Server 2008 Express and SQL Server Compact Edition – Veritabanı uygulamaları için kullanımı kolay ve güçlü bir araç

Visual Studio Express Editions yukarıdaki bileşenleri .NET platformu denilen tek bir platforma taşır.

.NET Framework hakkında aşağıda yeterli bilgi verilecektir.

Sıkça Sorulan Sorular

1. C# kaynak programını yazdım, derledim ve koşturdum. Bu eylemler diğer dillerde yaptıklarımdan farklı görünmüyor. Öyleyse .NET Framework ne işe yarıyor?

   .NET Framework'un mimarisi klâsik Windows uygulamalarına çok benzer. Bir .NET uygulamasını derlediğimizde ona ait .exe ya da .dll library dosyası ile birlikte AssemblyInfo dosyası oluşur. Bu dosya Win32 header, metadata, manifest ve MSIL kodlarını içerir. Biz .exe dosyasını çalıştırınca Win32 header dosyası CLR'i (Common Language Runtime) harekete geçirir, ona uygulamanın giriş noktasını gösterir. CLR o andan sonra JIT (Just-in-Time) derleyicisini ateşler. JIT, CLR 'dan gelen kodları işletim sisteminin anladığı makina diline dönüştürür. Görüldüğü gibi, biz bir programı derleyip koştururken .NET 'in (Framework, CLR, JIT, MSIL) bileşenleri devrededir, ancak onlar perdenin arkasındadırlar, biz görmeden rollerini oynarlar.

2. `Console` sınıfına ait bir nesne yaratmadığım halde, o sınıfın `Write()`, `WriteLine()`, `Read()`, `ReadLine()` metotlarını nasıl kullanabiliyorum?

   Bu metotlar `System.Console` sınıfının statik öğeleridir. Bir sınıfa ait nesne yaratılmadan o sınıfa ait statik öğeler kullanılabilir. Bu kavram ilerideki konularda ayrıntılı olarak açıklanacaktır.

3. Bir programda birden çok sınıf varsa, derleyici, `Main()` metodunun hangi sınıfta olduğunu nasıl biliyor? Birden çok sınıfta `Main()` metodu varsa, hangisini seçiyor?

   `Main()` metodunun hangi sınıfta olduğunu derleyiciye biz bildiriyoruz. Örneğin, ProgramDeneme.cs adlı programda `Main()` metodu Uygulama adlı sınıfın içindeyse ve programı satır komutuyla derliyorsak,

   ```
   csc ProgramDeneme.cs /main:Uygulama
   ```

   komutunu veririz. Eğer Visual Studio'yu kullanıyorsak

   Solution Explorer -> Properties -> Startup

   sekmesinden giriş noktası (Main() metodunu içeren sınıf) seçilebilir.

4. Bir sınıfın içine birden çok Main() metodu konulabilir mi?

   Hayır. Bir sınıfın içinde aynı adı taşıyan birden çok öğe olamaz. [Aşkın (overloaded) kavramını ileride ele alacağız.]

5. Komut satırı ile derleme ve koşturma eylemlerini ayrı ayrı yapabiliyoruz. Visual Studio'da programı derleme ve koşturma eylemlerini ayrı ayrı yapabilir miyiz?

   Evet. Programı derlemek için

   Build -> Build Solution

sekmesine tıklarız. Aynı işi F6 tuşu ile ya da Ctrl+Shift+B tuşlarıyla da yapabiliriz. Derlenen programı koşturmak için Debug -> Start Without Debug sekmesine tıklarız. Program önceden derlenmemişse, bu komut önce derleme eylemini sonra koşturma eylemini yapar. Aynı işi Ctrl+F5 tuşlarına basarak da yapabiliriz.

6. Visual Studio Express Editions hakkında nerelerde bilgi bulabilirim?

   Microsoft'un http://www.microsoft.com/express/ adresinde en son sistematik bilgileri ve hemen her konuyla ilgili öğretici dökümanları bulabilirsiniz. Ayrıca Visual Studio Express Editions kullanıcısı olarak kaydınızı yaptırabilirsiniz. Böylece bir çok programı ücretsiz edinebilirsiniz. Bu bağlamda

   ```
   MSDN 2008 Express Edition Library
   ```

   herkes için mükemmel bir kaynaktır. Ayrıca

   http://www.microsoft.com/express/vcsharp/

   adresinden her zaman bilgi alabilirsiniz.

7. *Visual Studio Express Editions* ile ticari programlar yazabilir miyim?

   Evet, yazabilirsiniz. Bunu yaptığınız için kimseye telif veya lisans ücreti ödemek zorunda değilsiniz.

8. Visual Studio Express Editions ile Visual Studio arasındaki fark nedir?

   Visual Studio profesyonel programcılar için daha çok materyale sahiptir. Visual Studio Express Editions yeni başlayanlar içindir. Expressi öğrenenler, isterlerse Visual Studio'ya yükselebilirler (upgade). Express ile yazılan programlar, kolayca Visual Studio ortamına taşınabilir. Tabii, Visual Studio'nun ücretli olduğunu söylemeye gerek yok.